Лабораторная работа № 1 по курсу дискретного анализа: сортировка за линейное время

Выполнил студент группы М80-208Б-22 МАИ *Караев Тариель*.

Условие

Кратко описывается задача:

1. Требуется разработать программу, осуществляющую ввод пар "ключ-значение сортировку по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод получившейся последовательности.

2. Вариант задания:

   - Сортировка подсчётом.
   - Тип ключа: числа от 0 до 65535.
   - Тип значения: числа от 0 до $2^{64} - 1$.

Метод решения

Так как в условии не сказано, сколько конкретно пар "ключ-значение"будет подано в программу, необходимо реализовать динамический массив – он же вектор – способный менять свой размер для хранения данных.

Далее требуется реализовать сортировку подсчётом. Так как по условию сортировка должна быть устойчивой, а мы имеем дело не с простыми числами, а с парами "ключ-значение необходимо конвертировать счётчик в массив префиксных сумм. Это нужно, чтобы при создании массива-результата мы могли знать, в какой индекс необходимо класть пары "ключ-значение".

Описание программы

Разделение по файлам, описание основных типов данных и функций.

- Класс TVector. Реализованы конструкторы, основные методы, также перегружены некоторые операторы вроде индексатора.

```
template <typename T>
class TVector {
    T* data;
    size_t size, capacity;

  public:
```

```
    TVector();
    TVector(size_t initSize);
    TVector(const TVector& other);
    TVector(TVector&& other);
    ~TVector();

    bool isEmpty();
    void Resize(size_t newSize);
    void PushBack(const T& newElement);
    void popBack();

    T* Data() const;
    size_t Size() const;
    size_t Capacity() const;

    T& operator[](size_t index);
    const T& operator[](size_t index) const;

  private:
    static T* Allocate(size_t newCapacity);
    void Reallocate(size_t newCapacity);
    void Destroy(T *object);
};
```

- Структура TPair. Необходим для хранения пар "ключ-значение".

```
struct TPair {
    uint16_t key;
    uint64_t value;
};
```

- Функция main. Здесь реализован алгоритм сортировки подсчётом. Работа ведётся с счётчиком counter и массивом данных data.

```
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);

    const size_t counterSize = UINT16_MAX + 1;
    TVector<size_t> counter(counterSize);
    for (size_t i = 0; i < counterSize; ++i) {
        counter[i] = 0;
    }
```

```
    TVector<TPair> data;
    TPair pair;
    while (std::cin >> pair.key >> pair.value) {
        ++counter[pair.key];
        data.PushBack(pair);
    }

    for (size_t i = 1; i < counter.Size(); ++i) {
        counter[i] = counter[i - 1] + counter[i];
    }

    TVector<TPair> result(data.Size());
    for (size_t i = data.Size(); i--> 0; ) {
        uint16_t key = data[i].key;
        result[counter[key] - 1] = data[i];
        --counter[key];
    }

    for (size_t i = 0; i < result.Size(); ++i) {
        std::cout << result[i].key << '\t'
                  << result[i].value << '\n';
    }

    return 0;
}
```

Дневник отладки

Изначально предполагался иной алгоритм, в которым должны были использоваться двумерные векторы и, возможно, очереди. Однако, в виду нежелания заниматься реализацией дополнительных структур данных, было принято решение использовать алгоритм, подразумевающий только векторы.

Тест производительности

Померить время работы кода лабораторной и теста производительности на разных объемах входных данных. Сравнить результаты. Проверить, что рост времени работы при увеличении объема входных данных согласуется с заявленной сложностью.

Сортировка подсчётом работает за линейное время. Если обратить на код программы, то видно, что функция main состоит из нескольких последовательных циклов. Для большей наглядности приведём таблицу, в которой написанная сортировка сравнивается со стандартными функция языка C++.

Количество пар "ключ-значение"	Sort(), мс	std::sort(), мс	std::stable_sort(), мс
1000	1247	280	187
10000	2167	3633	2206
100000	6974	38296	28208
1000000	55033	425589	341039
10000000	619328	5108635	3903731

На больших объёмах входных данных (от ста тысяч до десяти миллионов) становится заметно, что время сортировки пропорционально количеству пар "ключ-значение". Причём написанная функция оказывается заметно быстрее стандартных функций языка C++, так как те используют алгоритмы с временной сложностью $n \log n$.

Ниже приведена программа benchmark.cpp, использовавшаяся для засечения времени работы функций:

```
bool cmp(const std::pair<size_t, size_t>& a,
         const std::pair<size_t, size_t>& b) {
    return a.first < b.first;
}

int main() {
    TVector<TPair> data;
    std::vector<std::pair<size_t, size_t>> benchmarkData;

    TPair pair;
    while (std::cin >> pair.key >> pair.value) {
        data.PushBack(pair);
        benchmarkData.push_back({pair.key, pair.value});
    }

    auto start1 = std::chrono::high_resolution_clock::now();
    Sort(data);
    auto finish1 = std::chrono::high_resolution_clock::now();

    auto start2 = std::chrono::high_resolution_clock::now();
    sort(benchmarkData.begin(), benchmarkData.end(), cmp);
    auto finish2 = std::chrono::high_resolution_clock::now();

    auto start3 = std::chrono::high_resolution_clock::now();
    stable_sort(benchmarkData.begin(), benchmarkData.end(), cmp);
    auto finish3 = std::chrono::high_resolution_clock::now();

    auto duration1 =
        std::chrono::duration_cast<std::chrono::microseconds>(finish1
                                                              - start1);
```

```
    auto duration2 =
        std::chrono::duration_cast<std::chrono::microseconds>(finish2
                                                              - start2);
    auto duration3 =
        std::chrono::duration_cast<std::chrono::microseconds>(finish3
                                                              - start3);
    std::cout << "my sort duration: " << duration1.count() << std::endl;
    std::cout << "std sort duration: " << duration2.count() << std::endl;
    std::cout << "std stable_sort duration: " << duration3.count()
              << std::endl;

    return 0;
}
```

Выводы

В ходе выполнения программы был реализован алгоритм сортировки подсчётом. Данная сортировка имеет ограничение, однако даёт преимущество в виде линейной скорости работы. На мой взгляд, в большинстве случаев ограничение не столь значительно. Что самое важное: данная сортировка довольна проста в понимании и реализации. Помимо этого также был получен опыт реализации шаблонных структур данных.